Journal of Fuzzy Extension and Applications

www.journal-fea.com

J. Fuzzy. Ext. Appl. Vol. 7, No. 2 (2026) 592–614.

Paper Type: Research Paper

Python based Tool for Neutrosophic Cubic Sets and Their Operations

C V Malavika[1], Kalaivani Chandran[1], Sofia Jennifer John[2,*], Florentin Smarandache[3]

[1] Department of Mathematics, Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam 603110, India; malavika2210770@ssn.edu.in; kalaivanic@ssn.edu.in.

[2] Department of Information Technology, Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam 603110, India; sofiajenniferj@ssn.edu.in.

[3] Department of Mathematics and Science, University of New Mexico, Gallup, NM 87301, USA; smarand@unm.edu.

Citation:

Received: 14 November 2024 *Revised: 06 October 2025* *Accepted: 16 November 2025*	Malavika, C. V., Chandran, K., John, S. J., & Smarandache, F. (2026). Python based tool for neutrosophic cubic sets and their operations. *Journal of fuzzy extension and applications, 7*(2), 592-614.

Abstract

In recent years, Neutrosophic Cubic Sets have emerged as an advanced mathematical framework for representing various forms of uncertainty, indeterminacy, and inconsistency in complex systems. Although earlier research on Neutrosophic Cubic Sets provides fundamental definitions and basic theoretical constructs, this work significantly extends the framework by introducing clear and rigorous definitions for additional operations such as addition, multiplication, and scalar multiplication, as well as for important measures including score, accuracy, and certainty. To bridge the gap between theoretical constructs and practical applications, a highly interactive, user-friendly Python-based computational tool has been developed for performing these operations efficiently on Neutrosophic Cubic Sets. The usage of the tool is illustrated with examples, and the effectiveness of all the operations is demonstrated through images, thereby highlighting their potential utility for practical tasks such as image enhancement, pattern recognition and other applications where handling uncertain or inconsistent information is critical.

Keywords: Fuzzy set, Interval valued fuzzy set, Intuitionistic fuzzy set, Neutrosophic set, Interval-valued Neutrosophic set, Cubic set, Neutrosophic Cubic number, Neutrosophic Cubic set.

1|Introduction

1.1|Practical and Methodological Aims of the Study

Uncertainty modeling plays a central role in modern computational intelligence, particularly in areas such as decision-making, pattern recognition, and medical image processing. Classical fuzzy sets and their interval-valued extensions have provided strong tools for handling vagueness but remain insufficient when it comes to capturing indeterminacy and inconsistency simultaneously. Neutrosophic Cubic Sets, address these limitations by unifying the strengths of Single-Valued and Interval-Valued Neutrosophic Sets into a powerful hybrid structure. However, a key challenge lies in the absence of robust, standardized computational tools that allow for the real-world implementation and exploration of Neutrosophic Cubic Sets. This work aims to fill that gap by offering a

Corresponding Author: sofiajenniferj@ssn.edu.in
https://doi.org/10.22105/jfea.2025.488662.1702

Licensee System Analytics. This article is an open-access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0).

rigorously defined and practically implementable Python-based framework for performing core operations on Neutrosophic Cubic Sets.

1.2|Motivation for Conducting Research

While the literature presented in the upcoming section highlights significant theoretical advancements in neutrosophic logic, there is a clear lack of comprehensive software environments that make these theories usable for researchers and practitioners. Also the exisiting works related to Neutrosophic Cubic Sets lack the definitions for scalar multiplication, score, accuracy, and certainty - crucial for tasks like image analysis and uncertainty quantification. Without an accessible and validated computational platform, reproducing theoretical results, performing comparative analysis, or integrating Neutrosophic Cubic Sets into real-world applications becomes increasingly difficult. This gap between theoretical development and practical deployment forms the core motivation for this research.

1.3|Contributions of the Study

This study introduces a Python-based computational tool that operationalizes the complete framework of Neutrosophic Cubic Sets and Numbers. The key contributions are summarized as follows:

- Implementation of all formally defined operations including P-Union, R-Union, P-Intersection, R-Intersection, addition, multiplication, scalar multiplication, complement, containment, score, accuracy, and certainty.
- An interactive and intuitive user interface that allows real-time experimentation with Neutrosophic Cubic Sets.
- Demonstration of Neutrosophic Cubic Set operations in image processing and pattern recognition, particularly emphasizing their utility in managing noise, handling uncertainty, and enhancing local structural features.
- Empirical discussion on runtime performance, usability, and future extensions aimed at expanding applicability to domains such as medical diagnostics and decision support.

1.4|Organization of the Paper

The structure of the paper is organized as follows: Section 2 presents a comprehensive literature review, tracing the development from classical fuzzy sets to Neutrosophic Cubic Sets, while also identifying key research gaps. Section 3 introduces foundational definitions and mathematical preliminaries necessary for understanding Neutrosophic Cubic Sets. Section 4 outlines the complete Python implementation of Neutrosophic Cubic Numbers and Sets, detailing the supported operations and corresponding code structure. Section 5 describes the design and functionalities of the interactive Python-based tool, including its user interface, execution flow, and runtime efficiency. Section 6 illustrates the practical applicability of the tool in the context of image enhancement, showcasing how Neutrosophic Cubic Sets can be used for uncertainty-based visual processing. Section 7 discusses on some possible limitations of the proposed tool. Finally, Section 8 concludes the paper by summarizing the contributions and discussing potential directions for future research.

2|Literature Survey

Lotfi A. Zadeh in 1965 [12], independently introduced the fuzzy sets which uses one real value $\mu_{\mathtt{A}}(x) \in [0,1]$. It was used to represent the degree of certainty of different statements. Peng et al. [13] then presented an application in multi-criteria decision making problems after Zadeh. But sometimes, it is difficult to define $\mu_{\mathtt{A}}(x)$ with a single crisp value. Thus the idea of Interval-valued fuzzy set was introduced [14]. Interval-valued fuzzy set uses an interval value $[\mu_{\mathtt{AL}}(x), \mu_{\mathtt{AU}}(x)]$ with $0 \leq \mu_{\mathtt{AL}}(x) \leq \mu_{\mathtt{AU}}(x) \leq 1$.

Türkşen [14] extended the fuzzy set to an Interval-valued fuzzy set, which has many applications in Medical [5,6] and in Interval-valued logic (given by Türkşen) [7,8]. Atanassov (1986) [9], observed that fuzzy sets did not handle some kind of uncertainty in the data and thus he proposed the Intuitionistic Fuzzy sets, which became the generalisation of Fuzzy sets. Application of Intuitionistic fuzzy sets has been successful in several fields - Modeling Imprecision, Decision making problems, Pattern Recognition, Economics, Computational Intelligence, Medical Diagnosis. Further, Jun et al. [15] introduced fuzzy cubic set which is the combination of a Fuzzy set and an Interval-valued fuzzy set.

The real breakthrough in representing indeterminacy and inconsistency came with Florentin Smarandache's theory of Neutrosophy (1995) [11]. Neutrosophic sets have got an important role for denoising, clustering, segmentation, and classification methods in a number of medical image-processing applications.

The Neutrosophic set theory is a new mathematical tool for handling problems involving imprecise, indeterminacy, and inconsistent data. Neutrosophy provides the foundation for new mathematical theories. In a Neutrosophic set, an element has three associated defining elements - Truth(T), Indeterminacy (I) and Fallacy (F) defined on a universe of discourse X where the values of $T, I, F \in [1, 0]$. These three elements are completely independent. Neutrosophic set, basically studies the origin, nature and scope of neutralities and deals with the degree of uncertainty. Interval-valued Neutrosophic set [16] can represent uncertain, imprecise, incomplete and inconsistent information that exist in the real world. Interval-valued Neutrosophic sets generalises the Classical set, Fuzzy set, Interval-valued fuzzy set, Intuitionistic Fuzzy set, Interval-valued intuitionistic fuzzy set and the paraconsistent set.

Ali et al. (2016) introduced Neutrosophic Cubic Sets by combining the power of interval-valued and single-valued neutrosophic sets into a unified framework. This advancement was crucial for fields like pattern recognition and decision-making where uncertainty plays a central role [3]. Soon after, Jun et al. (2017) and Banerjee et al. (2017) explored the structure and application of NCS and Neutrosophic Cubic Environments (NCE) in decision-making problems [2,6].

To bring these theoretical constructs closer to application, El-Ghareeb (2019) proposed a general-purpose Python package for neutrosophic operations, marking a shift toward computational neutrosophy [4]. Nordo et al. (2024) and ShaliniTharanya et al. (2024) expanded this vision by exploring spherical, hyper, and extended neutrosophic models [5,7].

Recent literature demonstrates extensive usage of neutrosophic sets in real-world domains such as medical imaging [20,21,25], video analysis [23], social perception modeling [19], machine learning techniques for the diagnosis and detection of COVID-19 using AI models [24] and air quality monitoring [10]. The diversity of applications underscores the flexibility and potential of neutrosophic theory. Additionally, emerging models such as Stable Plithogenic Cubic Sets [18], Linguistic Neutrosophic Cubic Sets [22] have further enriched the neutrosophic framework.

Despite substantial theoretical progress in Neutrosophic Cubic Sets, the survey of existing literature highlights a lack of a comprehensive computational framework that integrates all essential operations formally defined in the domain, such as scalar multiplication, containment, score, accuracy, and certainty crucial for certain real-world applications. This gap becomes even more evident in the context of image processing - an area where Neutrosophic Cubic Sets have strong potential due to its ability to handle indeterminacy and noise. Addressing this need, the proposed work introduces an interactive Python-based tool that not only supports all fundamental and extended Neutrosophic Cubic Set operations but also showcases its role in image enhancement, thus bridging both theoretical and application-level gaps.

3|Preliminaries

This section introduces the key definitions that form the basis for the discussions and results presented in this paper.

Definition 1. *[1] Let X be a non-empty set and $\mathtt{I} = [0, 1]$. A Neutrosophic set A on X is a mapping defined as*

$$A = \langle T_{\mathtt{A}}, I_{\mathtt{A}}, F_{\mathtt{A}} \rangle : X \longrightarrow \zeta$$

where $\zeta = \mathtt{I}^3$ and $T_{\mathtt{A}}, I_{\mathtt{A}}, F_{\mathtt{A}} : \mathtt{X} \longrightarrow \mathtt{I}$ such that $0 \leq T_{\mathtt{A}} + I_{\mathtt{A}} + F_{\mathtt{A}} \leq 3$.

Definition 2. *[1] The Interval-valued Neutrosophic set A is of the form*

$$A = \left\langle [T_{\mathtt{AL}}, T_{\mathtt{AU}}], [I_{\mathtt{AL}}, I_{\mathtt{AU}}], [F_{\mathtt{AL}}, F_{\mathtt{AU}}] \right\rangle$$

where $T_{\mathtt{AL}}, T_{\mathtt{AU}}, I_{\mathtt{AL}}, I_{\mathtt{AU}}, F_{\mathtt{AL}}, F_{\mathtt{AU}} : X \longrightarrow [0,1]$ *such that* $T_{\mathtt{AL}} \leq T_{\mathtt{AU}}, I_{\mathtt{AL}} \leq I_{\mathtt{AU}}, F_{\mathtt{AL}} \leq F_{\mathtt{AU}}, 0 \leq T_{\mathtt{AL}} + I_{\mathtt{AL}} + F_{\mathtt{AL}} \leq 3$ *and* $0 \leq T_{\mathtt{AU}} + I_{\mathtt{AU}} + F_{\mathtt{AU}} \leq 3$.

The following Python implementation is a slightly modified version of the implementation for Single-Valued Neutrosophic Numbers (SVNNs) and Interval-Valued Neutrosophic Numbers (IVNNs) in the text [5, 17].

In Python, the constructor method ___*init*___ is automatically invoked when a new instance of a class is created. The class definitions for Neutrosophic Set (NS) and Interval-Valued Neutrosophic Set (INS), including their respective constructors, are provided below.

```
class NS:

    def __init__(self,
                 truth,
                 indeterminacy,
                 falsehood):

        assert 0 <= truth <= 1
        assert 0 <= indeterminacy <= 1
        assert 0 <= falsehood <= 1
        assert 0 <= truth + indeterminacy + falsehood <= 3

        self.T = truth
        self.I = indeterminacy
        self.F = falsehood
```

```
class INS:

    def __init__(self,
                 T_lower, T_upper,
                 I_lower, I_upper,
                 F_lower, F_upper):

        assert 0 <= T_lower <= 1
        assert 0 <= T_upper <= 1
        assert 0 <= I_lower <= 1
        assert 0 <= I_upper <= 1
        assert 0 <= F_lower <= 1
        assert 0 <= F_upper <= 1
        assert T_lower <= T_upper
        assert I_lower <= I_upper
        assert F_lower <= F_upper
        assert 0 <= T_lower + I_lower + F_lower <= 3
        assert 0 <= T_upper + I_upper + F_upper <= 3

        self.T_lower = T_lower
        self.T_upper = T_upper
        self.I_lower = I_lower
        self.I_upper = I_upper
        self.F_lower = F_lower
        self.F_upper = F_upper
```

An additional method is implemented to provide the string representation of a Neutrosophic Set and an Interval-Valued Neutrosophic Set, respectively, by overriding the ___*str*___ method in Python.

NS class:

```
def __str__(self):

    return (f"<{self.T},"
```

```
            f"{self.I},"
            f"{self.F}>")
```

INS class:

```
def __str__(self):

    return (f"<[{self.T_lower}, {self.T_upper}],"
            f"[{self.I_lower},{self.I_upper}],"
            f"[{self.F_lower},{self.F_upper}]>")
```

4|Python implementation for Neutrosophic Cubic set

The Python implementation of the Neutrosophic Cubic Number, the Neutrosophic Cubic Set, and their basic operations is provided in this section.

Definition 3. *[3] The Neutrosophic Cubic Set is denoted as $\langle A, \lambda\rangle$ and is defined as*

$$\langle A, \lambda\rangle \quad = \quad \Big\{\langle A_{\mathtt{x}}, \lambda_{\mathtt{x}}\rangle : x \in X\Big\}$$

where

$$A_{\mathtt{x}} \quad = \quad \Big\langle [T_{\mathtt{AL}}(x), T_{\mathtt{AU}}(x)], [I_{\mathtt{AL}}(x), I_{\mathtt{AU}}(x)], [F_{\mathtt{AL}}(x), F_{\mathtt{AU}}(x)]\Big\rangle$$

and

$$\lambda_{\mathtt{x}} \quad = \quad \Big\langle T_{\lambda}(x), I_{\lambda}(x), F_{\lambda}(x)\Big\rangle$$

Here, $\langle A_{\mathtt{x}}, \lambda_{\mathtt{x}}\rangle$ is called as Neutrosophic Cubic Number.

Given below is the constructor function, which is automatically invoked when a new object of the *NCN* (Neutrosophic Cubic Number) class is instantiated.

```
def __init__(self,
    T_lower, T_upper,
    I_lower, I_upper,
    F_lower, F_upper,
    truth,
    indeterminacy,
    falsehood):

    self.ins = INS(T_lower, T_upper, I_lower, I_upper, F_lower, F_upper)
    self.ns = NS(truth, indeterminacy, falsehood)
```

Definition 4. *[3] Let $\mathtt{X}$ be a non-empty set. Then the P-Union of two Neutrosophic Cubic Sets $\langle A, \lambda\rangle$ and $\langle B, \mu\rangle$ is defined as*

$$\langle A \cup_{\mathtt{P}} B, \lambda \cup_{\mathtt{P}} \mu\rangle \quad = \quad \Big\{\langle A_{\mathtt{x}} \cup_{\mathtt{P}} B_{\mathtt{x}}, \lambda_{\mathtt{x}} \cup_{\mathtt{P}} \mu_{\mathtt{x}}\rangle : x \in X\Big\}$$

where

$$\begin{aligned} A_{\mathtt{x}} \cup_{\mathtt{P}} B_{\mathtt{x}} = \Big\langle & [T_{\mathtt{AL}}(x) \vee T_{\mathtt{BL}}(x), T_{\mathtt{AU}}(x) \vee T_{\mathtt{BU}}(x)], [I_{\mathtt{AL}}(x) \vee I_{\mathtt{BL}}(x), I_{\mathtt{AU}}(x) \vee I_{\mathtt{BU}}(x)], \\ & [F_{\mathtt{AL}}(x) \vee F_{\mathtt{BL}}(x), F_{\mathtt{AU}}(x) \vee F_{\mathtt{BU}}(x)]\Big\rangle \end{aligned}$$

and

$$\lambda_{\mathtt{x}} \cup_{\mathtt{P}} \mu_{\mathtt{x}} \quad = \quad \Big\langle T_{\lambda}(x) \vee T_{\mu}(x), I_{\lambda}(x) \vee I_{\mu}(x), F_{\lambda}(x) \vee F_{\mu}(x)\Big\rangle$$

Here, $\langle A_{\mathtt{x}} \cup_{\mathtt{P}} B_{\mathtt{x}}, \lambda_{\mathtt{x}} \cup_{\mathtt{P}} \mu_{\mathtt{x}}\rangle$ is called the P-Union of the two Neutrosophic Cubic Numbers.

The following code computes the P-Union of two Neutrosophic Cubic Numbers.

```
def p_union(self, other):

    return NCN(
        max(self.ins.T_lower, other.ins.T_lower),
        max(self.ins.T_upper, other.ins.T_upper),
        max(self.ins.I_lower, other.ins.I_lower),
        max(self.ins.I_upper, other.ins.I_upper),
        max(self.ins.F_lower, other.ins.F_lower),
        max(self.ins.F_upper, other.ins.F_upper),
        max(self.ns.T, other.ns.T),
        max(self.ns.I, other.ns.I),
        max(self.ns.F, other.ns.F)
    )
```

Definition 5. *[3] Let* $\mathtt{X}$ *be a non-empty set. Then the P-Intersection of two Neutrosophic Cubic Sets* $\langle A, \lambda\rangle$ *and* $\langle B, \mu\rangle$ *is defined as*

$$\langle A \cap_{\mathtt{P}} B, \lambda \cap_{\mathtt{P}} \mu\rangle \quad = \quad \Big\{\langle A_{\mathtt{x}} \cap_{\mathtt{P}} B_{\mathtt{x}}, \lambda_{\mathtt{x}} \cap_{\mathtt{P}} \mu_{\mathtt{x}}\rangle : x \in X\Big\}$$

where

$$A_{\mathtt{x}} \cap_{\mathtt{P}} B_{\mathtt{x}} = \Big\langle [T_{\mathtt{AL}}(x) \wedge T_{\mathtt{BL}}(x), T_{\mathtt{AU}}(x) \wedge T_{\mathtt{BU}}(x)], [I_{\mathtt{AL}}(x) \wedge I_{\mathtt{BL}}(x), I_{\mathtt{AU}}(x) \wedge I_{\mathtt{BU}}(x)],$$
$$[F_{\mathtt{AL}}(x) \wedge F_{\mathtt{BL}}(x), F_{\mathtt{AU}}(x) \wedge F_{\mathtt{BU}}(x)]\Big\rangle$$

and

$$\lambda_{\mathtt{x}} \cap_{\mathtt{P}} \mu_{\mathtt{x}} \quad = \quad \Big\langle T_{\lambda}(x) \wedge T_{\mu}(x), I_{\lambda}(x) \wedge I_{\mu}(x), F_{\lambda}(x) \wedge F_{\mu}(x)\Big\rangle$$

Here, $\langle A_{\mathtt{x}} \cap_{\mathtt{P}} B_{\mathtt{x}}, \lambda_{\mathtt{x}} \cap_{\mathtt{P}} \mu_{\mathtt{x}}\rangle$ *is called the P-Intersection of the two Neutrosophic Cubic Numbers.*

The P-Intersection operation for two Neutrosophic Cubic Numbers is implemented as follows.

```
def p_intersection(self, other):

    return NCN(
        min(self.ins.T_lower, other.ins.T_lower),
        min(self.ins.T_upper, other.ins.T_upper),
        min(self.ins.I_lower, other.ins.I_lower),
        min(self.ins.I_upper, other.ins.I_upper),
        min(self.ins.F_lower, other.ins.F_lower),
        min(self.ins.F_upper, other.ins.F_upper),
        min(self.ns.T, other.ns.T),
        min(self.ns.I, other.ns.I),
        min(self.ns.F, other.ns.F)
    )
```

Definition 6. *[3] Let* $\mathtt{X}$ *be a non-empty set. Then the R-Union of two Neutrosophic Cubic Sets* $\langle A, \lambda\rangle$ *and* $\langle B, \mu\rangle$ *is defined as*

$$\langle A \cup_{\mathtt{R}} B, \lambda \cup_{\mathtt{R}} \mu\rangle \quad = \quad \Big\{\langle A_{\mathtt{x}} \cup_{\mathtt{R}} B_{\mathtt{x}}, \lambda_{\mathtt{x}} \cup_{\mathtt{R}} \mu_{\mathtt{x}}\rangle : x \in X\Big\}$$

where

$$A_{\mathtt{x}} \cup_{\mathtt{R}} B_{\mathtt{x}} = \Big\langle [T_{\mathtt{AL}}(x) \vee T_{\mathtt{BL}}(x), T_{\mathtt{AU}}(x) \vee T_{\mathtt{BU}}(x)], [I_{\mathtt{AL}}(x) \vee I_{\mathtt{BL}}(x), I_{\mathtt{AU}}(x) \vee I_{\mathtt{BU}}(x)],$$
$$[F_{\mathtt{AL}}(x) \vee F_{\mathtt{BL}}(x), F_{\mathtt{AU}}(x) \vee F_{\mathtt{BU}}(x)]\Big\rangle$$

and

$$\lambda_{\mathtt{x}} \cup_{\mathtt{R}} \mu_{\mathtt{x}} \quad = \quad \Big\langle T_{\lambda}(x) \wedge T_{\mu}(x), I_{\lambda}(x) \wedge I_{\mu}(x), F_{\lambda}(x) \wedge F_{\mu}(x)\Big\rangle$$

Here, $\langle A_{\mathtt{x}} \cup_{\mathtt{R}} B_{\mathtt{x}}, \lambda_{\mathtt{x}} \cup_{\mathtt{P}} \mu_{\mathtt{x}}\rangle$ *is called the R-Union of the two Neutrosophic Cubic Numbers.*

The following snippet demonstrates the R-Union of two Neutrosophic Cubic Numbers.

```
def r_union(self, other):

    return NCN(
        max(self.ins.T_lower, other.ins.T_lower),
        max(self.ins.T_upper, other.ins.T_upper),
        max(self.ins.I_lower, other.ins.I_lower),
        max(self.ins.I_upper, other.ins.I_upper),
        max(self.ins.F_lower, other.ins.F_lower),
        max(self.ins.F_upper, other.ins.F_upper),
        min(self.ns.T, other.ns.T),
        min(self.ns.I, other.ns.I),
        min(self.ns.F, other.ns.F)
    )
```

Definition 7. *[3] Let* $\mathtt{X}$ *be a non-empty set. Then the R-Intersection of two Neutrosophic Cubic Sets* $\langle A, \lambda\rangle$ *and* $\langle B, \mu\rangle$ *is defined as*

$$\langle A \cap_{\mathtt{R}} B, \lambda \cap_{\mathtt{R}} \mu\rangle \quad = \quad \Big\{\langle A_{\mathtt{x}} \cap_{\mathtt{R}} B_{\mathtt{x}}, \lambda_{\mathtt{x}} \cap_{\mathtt{R}} \mu_{\mathtt{x}}\rangle : x \in X\Big\}$$

where

$$A_{\mathtt{x}} \cap_{\mathtt{R}} B_{\mathtt{x}} = \Big\langle [T_{\mathtt{AL}}(x) \wedge T_{\mathtt{BL}}(x), T_{\mathtt{AU}}(x) \wedge T_{\mathtt{BU}}(x)], [I_{\mathtt{AL}}(x) \wedge I_{\mathtt{BL}}(x), I_{\mathtt{AU}}(x) \wedge I_{\mathtt{BU}}(x)], \\ [F_{\mathtt{AL}}(x) \wedge F_{\mathtt{BL}}(x), F_{\mathtt{AU}}(x) \wedge F_{\mathtt{BU}}(x)]\Big\rangle$$

and

$$\lambda_{\mathtt{x}} \cap_{\mathtt{R}} \mu_{\mathtt{x}} \quad = \quad \Big\langle T_{\lambda}(x) \vee T_{\mu}(x), I_{\lambda}(x) \vee I_{\mu}(x), F_{\lambda}(x) \vee F_{\mu}(x)\Big\rangle$$

Here, $\langle A_{\mathtt{x}} \cap_{\mathtt{R}} B_{\mathtt{x}}, \lambda_{\mathtt{x}} \cap_{\mathtt{R}} \mu_{\mathtt{x}}\rangle$ *is called the R-Intersection of the two Neutrosophic Cubic Numbers.*

Below is the implementation for computing the R-Intersection of two Neutrosophic Cubic Numbers.

```
def r_intersection(self, other):

    return NCN(
        min(self.ins.T_lower, other.ins.T_lower),
        min(self.ins.T_upper, other.ins.T_upper),
        min(self.ins.I_lower, other.ins.I_lower),
        min(self.ins.I_upper, other.ins.I_upper),
        min(self.ins.F_lower, other.ins.F_lower),
        min(self.ins.F_upper, other.ins.F_upper),
        max(self.ns.T, other.ns.T),
        max(self.ns.I, other.ns.I),
        max(self.ns.F, other.ns.F)
    )
```

Definition 8. *[3] Let* $\mathtt{X}$ *be a non-empty set. Then the Complement of a Neutrosophic Cubic Sets* $\langle A, \lambda\rangle$ *is defined as*

$$\langle A^c, \lambda^c\rangle \quad = \quad \Big\{\langle A^c_{\mathtt{x}}, \lambda^c_{\mathtt{x}}\rangle : x \in X\Big\}$$

where

$$A^c_{\mathtt{x}} \quad = \quad \Big\langle [1 - T_{\mathtt{AL}}(x), 1 - T_{\mathtt{AU}}(x)], [1 - I_{\mathtt{AL}}(x), 1 - I_{\mathtt{AU}}(x)], [1 - F_{\mathtt{AL}}(x), 1 - F_{\mathtt{AU}}(x)]\Big\rangle$$

and

$$\lambda^c_{\mathtt{x}} \quad = \quad \Big\langle 1 - T_{\lambda}(x), 1 - I_{\lambda}(x), 1 - F_{\lambda}(x)\Big\rangle$$

Here, $\langle A^c_{\mathtt{x}}, \lambda^c_{\mathtt{x}}\rangle$ *is called the Complement of a Neutrosophic Cubic Number.*

Below is the implementation for computing the Complement of a Neutrosophic Cubic Number.

```
def complement(self):

    return NCN(
        round(1 - self.ins.T_upper, 2),
        round(1 - self.ins.T_lower, 2),
        round(1 - self.ins.I_upper, 2),
        round(1 - self.ins.I_lower, 2),
        round(1 - self.ins.F_upper, 2),
        round(1 - self.ins.F_lower, 2),
        round(1 - self.ns.T, 2),
        round(1 - self.ns.I, 2),
        round(1 - self.ns.F, 2)
    )
```

Definition 9. *[3] Let* $\mathtt{X}$ *be a non-empty set. Then two Neutrosophic Cubic Sets* $\langle A, \lambda\rangle$ *and* $\langle B, \mu\rangle$ *are said to be equal if and only if*

$T_{\mathtt{AL}}(x) = T_{\mathtt{BL}}(x)$, $T_{\mathtt{AU}}(x) = T_{\mathtt{BU}}(x)$, $I_{\mathtt{AL}}(x) = I_{\mathtt{BL}}(x)$, $I_{\mathtt{AU}}(x) = I_{\mathtt{BU}}(x)$, $F_{\mathtt{AL}}(x) = F_{\mathtt{BL}}(x)$, $F_{\mathtt{AU}}(x) = F_{\mathtt{BU}}(x)$, $T_\lambda(x) = T_\mu(x)$, $I_\lambda(x) = I_\mu(x)$ *and* $F_\lambda(x) = F_\mu(x)$
and is denoted as

$$\langle A, \lambda\rangle \quad = \quad \langle B, \mu\rangle$$

This code tests the equality of two Neutrosophic Cubic Numbers by overriding the ___*eq*___ method in Python.

```
def __eq__(self, other):

    return (
        self.ins.T_lower == other.ins.T_lower and
        self.ins.T_upper == other.ins.T_upper and
        self.ins.I_lower == other.ins.I_lower and
        self.ins.I_upper == other.ins.I_upper and
        self.ins.F_lower == other.ins.F_lower and
        self.ins.F_upper == other.ins.F_upper and
        self.ns.T == other.ns.T and
        self.ns.I == other.ns.I and
        self.ns.F == other.ns.F
    )
```

Definition 10. *Let* $\mathtt{X}$ *be a non-empty set. Then the Addition of two Neutrosophic Cubic Sets* $\langle A, \lambda\rangle$ *and* $\langle B, \mu\rangle$ *is defined as*

$$\langle A + B, \lambda + \mu\rangle \quad = \quad \Big\{\langle A_{\mathtt{x}} + B_{\mathtt{x}}, \lambda_{\mathtt{x}} + \mu_{\mathtt{x}}\rangle : x \in X\Big\}$$

where

$$A_{\mathtt{x}} + B_{\mathtt{x}} = \Big\langle [T_{\mathtt{AL}}(x) + T_{\mathtt{BL}}(x) - T_{\mathtt{AL}}(x) * T_{\mathtt{BL}}(x), T_{\mathtt{AU}}(x) + T_{\mathtt{BU}}(x) - T_{\mathtt{AU}}(x) * T_{\mathtt{BU}}(x)],$$
$$[I_{\mathtt{AL}}(x) * I_{\mathtt{BL}}(x), I_{\mathtt{AU}}(x) * I_{\mathtt{BU}}(x)], [F_{\mathtt{AL}}(x) * F_{\mathtt{BL}}(x), F_{\mathtt{AU}}(x) * F_{\mathtt{BU}}(x)]\Big\rangle$$

and

$$\lambda_{\mathtt{x}} + \mu_{\mathtt{x}} \quad = \quad \Big\langle T_\lambda(x) + T_\mu(x) - T_\lambda(x) * T_\mu(x), I_\lambda(x) * I_\mu(x), F_\lambda(x) * F_\mu(x)\Big\rangle$$

Here, $\langle A_{\mathtt{x}} + B_{\mathtt{x}}, \lambda_{\mathtt{x}} + \mu_{\mathtt{x}}\rangle$ *is the result on Addition of two Neutrosophic Cubic Numbers.*

The implementation for the addition of two Neutrosophic Cubic Numbers is given below and is achieved by overriding the ___*add*___ method in Python.

```
def __add__(self, other):

    return NCN(
        round(self.ins.T_lower + other.ins.T_lower - self.ins.T_lower * other.ins.T_lower, 2),
        round(self.ins.T_upper + other.ins.T_upper - self.ins.T_upper * other.ins.T_upper, 2),
        round(self.ins.I_lower * other.ins.I_lower, 2),
        round(self.ins.I_upper * other.ins.I_upper, 2),
        round(self.ins.F_lower * other.ins.F_lower, 2),
        round(self.ins.F_upper * other.ins.F_upper, 2),
        round(self.ns.T + other.ns.T - self.ns.T * other.ns.T, 2),
        round(self.ns.I * other.ns.I, 2),
        round(self.ns.F * other.ns.F, 2)
    )
```

Definition 11. *Let* X *be a non-empty set. Then the Multiplication of two Neutrosophic Cubic Sets* $\langle A, \lambda\rangle$ *and* $\langle B, \mu\rangle$ *is defined as*

$$\langle A * B, \lambda * \mu\rangle \quad = \quad \Big\{\langle A_{\mathtt{x}} * B_{\mathtt{x}}, \lambda_{\mathtt{x}} * \mu_{\mathtt{x}}\rangle : x \in X\Big\}$$

where

$$\begin{aligned} A_{\mathtt{x}} * B_{\mathtt{x}} = \Big\langle & [T_{\mathtt{AL}}(x) * T_{\mathtt{BL}}(x), T_{\mathtt{AU}}(x) * T_{\mathtt{BU}}(x)], \\ & [I_{\mathtt{AL}}(x) + I_{\mathtt{BL}}(x) - I_{\mathtt{AL}}(x) * I_{\mathtt{BL}}(x), I_{\mathtt{AU}}(x) + I_{\mathtt{BU}}(x) - I_{\mathtt{AU}}(x) * I_{\mathtt{BU}}(x)], \\ & [F_{\mathtt{AL}}(x) + F_{\mathtt{BL}}(x) - F_{\mathtt{AL}}(x) * F_{\mathtt{BL}}(x), F_{\mathtt{AU}}(x) + F_{\mathtt{BU}}(x) - F_{\mathtt{AU}}(x) * F_{\mathtt{BU}}(x)]\Big\rangle \end{aligned}$$

and

$$\begin{aligned} \lambda_{\mathtt{x}} * \mu_{\mathtt{x}} = \Big\langle & T_{\lambda}(x) * T_{\mu}(x), I_{\lambda}(x) + I_{\mu}(x) - I_{\lambda}(x) * I_{\mu}(x), \\ & F_{\lambda}(x) + F_{\mu}(x) - F_{\lambda}(x) * F_{\mu}(x)\Big\rangle \end{aligned}$$

Here, $\langle A_{\mathtt{x}} * B_{\mathtt{x}}, \lambda_{\mathtt{x}} * \mu_{\mathtt{x}}\rangle$ *is the result on Multiplication of two Neutrosophic Cubic Numbers.*

The following code implements the multiplication of two Neutrosophic Cubic Numbers by overriding the *__mul__* method in Python.

```
def __mul__(self, other):

    return NCN(
        round(self.ins.T_lower * other.ins.T_lower, 2),
        round(self.ins.T_upper * other.ins.T_upper, 2),
        round(self.ins.I_lower + other.ins.I_lower - self.ins.I_lower * other.ins.I_lower, 2),
        round(self.ins.I_upper + other.ins.I_upper - self.ins.I_upper * other.ins.I_upper, 2),
        round(self.ins.F_lower + other.ins.F_lower - self.ins.F_lower * other.ins.F_lower, 2),
        round(self.ins.F_upper + other.ins.F_upper - self.ins.F_upper * other.ins.F_upper, 2),
        round(self.ns.T * other.ns.T, 2),
        round(self.ns.I + other.ns.I - self.ns.I * other.ns.I, 2),
        round(self.ns.F + other.ns.F - self.ns.F * other.ns.F, 2)
    )
```

Definition 12. *Let* X *be a non-empty set. Then the Scalar Multiplication a Neutrosophic Cubic Set* $\langle A, \lambda\rangle$ *with a positive scalar* k *is defined as*

$$\langle k * A, k * \lambda\rangle \quad = \quad \Big\{\langle k * A_{\mathtt{x}}, k * \lambda_{\mathtt{x}}\rangle : x \in X\Big\}$$

where

$$k * A_{\mathtt{x}} \quad = \quad \Big\langle [1 - (T_{\mathtt{AL}}(x))^k, 1 - (T_{\mathtt{AU}}(x))^k], [(I_{\mathtt{AL}}(x))^k, (I_{\mathtt{AU}}(x))^k], [(F_{\mathtt{AL}}(x))^k, (F_{\mathtt{AU}}(x))^k]\Big\rangle$$

and

$$k * \lambda_{\mathtt{x}} \quad = \quad \Big\langle 1 - (T_{\lambda}(x))^k, (I_{\lambda}(x))^k, (F_{\lambda}(x))^k\Big\rangle$$

Here, $\langle k * A_{\mathtt{x}}, k * \lambda_{\mathtt{x}} \rangle$ is the result after Scalar Multiplication of a positive scalar k with a Neutrosophic Cubic Number.

Below is the implementation for performing scalar multiplication.

```
def scalar_multiplication(self, scalar):

    assert scalar > 0

    return NCN(
        round(1 - ((1 - self.ins.T_lower) ** scalar), 2),
        round(1 - ((1 - self.ins.T_upper) ** scalar), 2),
        round(self.ins.I_lower ** scalar, 2),
        round(self.ins.I_upper ** scalar, 2),
        round(self.ins.F_lower ** scalar, 2),
        round(self.ins.F_upper ** scalar, 2),
        round(1 - ((1 - self.ns.T) ** scalar), 2),
        round(self.ns.I ** scalar, 2),
        round(self.ns.F ** scalar, 2)
    )
```

Definition 13. *[3] Let* $\mathtt{X}$ *be a non-empty set. Then a Neutrosophic Cubic Set $\langle A, \lambda \rangle$ is said to be a subset or contained in $\langle B, \mu \rangle$ if and only if*

$T_{\mathtt{AL}}(x) \leq T_{\mathtt{BL}}(x)$, $T_{\mathtt{AU}}(x) \leq T_{\mathtt{BU}}(x)$, $I_{\mathtt{AL}}(x) \geq I_{\mathtt{BL}}(x)$, $I_{\mathtt{AU}}(x) \geq I_{\mathtt{BU}}(x)$, $F_{\mathtt{AL}}(x) \geq F_{\mathtt{BL}}(x)$, $F_{\mathtt{AU}}(x) \geq F_{\mathtt{BU}}(x)$, $T_{\lambda}(x) \leq T_{\mu}(x)$, $I_{\lambda}(x) \geq I_{\mu}(x)$ *and* $F_{\lambda}(x) \geq F_{\mu}(x)$
and is denoted as

$$\langle A, \lambda \rangle \quad \subseteq \quad \langle B, \mu \rangle$$

This code checks whether one Neutrosophic Cubic Number is a subset of, or contained within, another.

```
def containment(self, other):

    return (
        self.ins.T_lower <= other.ins.T_lower and
        self.ins.T_upper <= other.ins.T_upper and
        self.ins.I_lower >= other.ins.I_lower and
        self.ins.I_upper >= other.ins.I_upper and
        self.ins.F_lower >= other.ins.F_lower and
        self.ins.F_upper >= other.ins.F_upper and
        self.ns.T <= other.ns.T and
        self.ns.I >= other.ns.I and
        self.ns.F >= other.ns.F
    )
```

Definition 14. *Let* $\mathtt{X}$ *be a non-empty set. Then the Score of a Neutrosophic Cubic Set $\langle A, \lambda \rangle$ is defined as*

$$(6 + T_{\mathtt{AL}}(x) + T_{\mathtt{AU}}(x) - I_{\mathtt{AL}}(x) - I_{\mathtt{AU}}(x) - F_{\mathtt{AL}}(x) - F_{\mathtt{AU}}(x) +$$
$$T_{\lambda}(x) - I_{\lambda}(x) - F_{\lambda}(x))/9 : x \in X$$

The following code computes the score of a Neutrosophic Cubic Number.

```
def score(self):

    return (
        round(((6 + self.ins.T_lower + self.ins.T_upper
        - self.ins.I_lower - self.ins.I_upper
        - self.ins.F_lower - self.ins.F_upper
        + self.ns.T - self.ns.I - self.ns.F) / 9), 2)
    )
```

Definition 15. *Let* X *be a non-empty set. Then the Accuracy of a Neutrosophic Cubic Set* $\langle A, \lambda \rangle$ *is defined as*

$$T_{\mathtt{AL}}(x) + T_{\mathtt{AU}}(x) - F_{\mathtt{AL}}(x) - F_{\mathtt{AU}}(x) + T_{\lambda}(x) - F_{\lambda}(x) : x \in X$$

The following snippet calculates the accuracy of a Neutrosophic Cubic Number.

```
def accuracy(self):

    return (
        round((self.ins.T_lower + self.ins.T_upper
        - self.ins.F_lower - self.ins.F_upper
        + self.ns.T - self.ns.F), 2)
    )
```

Definition 16. *Let* X *be a non-empty set. Then the Certainty of a Neutrosophic Cubic Set* $\langle A, \lambda \rangle$ *is defined as*

$$T_{\mathtt{AL}}(x) + T_{\mathtt{AU}}(x) + T_{\lambda}(x) : x \in X$$

Certainty for a Neutrosophic Cubic Number is calculated using the code snippet below.

```
def certainty(self):

    return (
        round((self.ins.T_lower + self.ins.T_upper + self.ns.T), 2)
    )
```

The following method provides the string representation of an NCN (Neutrosophic Cubic Number) by overriding the *__str__* method in Python.

```
def __str__(self):

    return (f"({str(self.ins)}, "
            f"{str(self.ns)})")
```

The implementation of the *__len__* and *__getitem__* methods in Python, which enable the *NCN* class to behave like an iterable, is given below.

```
def __len__(self):

    return len(self._number)
```

```
def __getitem__(self, index):

    return self._number[index]
```

Based on the Python implementation of the Neutrosophic Cubic Number (*NCN*) and its operations, the implementation for the Neutrosophic Cubic Set (*NCS*) and its operations is provided below.

```
class NCS:

    def __init__(self):

        self.ncs = []


    def add_element(self, ncn):

        assert isinstance(ncn, NCN)
        self.ncs.append(ncn)


    def __str__(self):

        string = "{\n"
```

```
        for i in range(len(self.ncs)):
            if i < len(self.ncs) - 1:
                string += f"{self.ncs[i]}, \n"
            else:
                string += f"{self.ncs[i]}"
        string += "\n}"

        return string


    def __len__(self):

        return len(self.ncs)


    def __getitem__(self, index):

        return self.ncs[index]


    def p_union(self, other):

        p_union_set = NCS()
        for i in range(len(self.ncs)):
            p_union_set.add_element(self.ncs[i].p_union(other.ncs[i]))

        return p_union_set


    def p_intersection(self, other):

        p_intersection_set = NCS()
        for i in range(len(self.ncs)):
            p_intersection_set.add_element(self.ncs[i].p_intersection(other.ncs[i]))

        return p_intersection_set


    def r_union(self, other):

        r_union_set = NCS()
        for i in range(len(self.ncs)):
            r_union_set.add_element(self.ncs[i].r_union(other.ncs[i]))

        return r_union_set


    def r_intersection(self, other):

        r_intersection_set = NCS()
        for i in range(len(self.ncs)):
            r_intersection_set.add_element(self.ncs[i].r_intersection(other.ncs[i]))

        return r_intersection_set


    def complement(self):

        complement_set = NCS()
        for i in range(len(self.ncs)):
            complement_set.add_element(self.ncs[i].complement())

        return complement_set

    def __eq__(self, other):
```

```
        for i in range(len(self.ncs)):
            if not (self.ncs[i] == other.ncs[i]):
                return False
        return True


    def __add__(self, other):

        add_set = NCS()
        for i in range(len(self.ncs)):
            add_set.add_element(self.ncs[i] + other.ncs[i])

        return add_set


    def __mul__(self, other):

        mul_set = NCS()
        for i in range(len(self.ncs)):
            mul_set.add_element(self.ncs[i] * other.ncs[i])

        return mul_set


    def scalar_multiplication(self, scalar):

        scalar_mul_set = NCS()
        for i in range(len(self.ncs)):
            scalar_mul_set.add_element(self.ncs[i].scalar_multiplication(scalar))

        return scalar_mul_set


    def containment(self, other):

        for i in range(len(self.ncs)):
            if not(self.ncs[i].containment(other.ncs[i])):
                return False
        return True


    def score(self):

        score_set = []
        for i in range(len(self.ncs)):
            score_set.append(self.ncs[i].score())

        return score_set


    def accuracy(self):

        accuracy_set = []
        for i in range(len(self.ncs)):
            accuracy_set.append(self.ncs[i].accuracy())

        return accuracy_set


    def certainty(self):

        certainty_set = []
        for i in range(len(self.ncs)):
            certainty_set.append(self.ncs[i].certainty())
```

```
    return certainty_set
```

5|Python tool for Neutrosophic Cubic Sets

5.1|Tool Overview

The developed Python-based tool provides an intuitive and interactive environment for defining, manipulating, and analyzing Neutrosophic Cubic Sets. It enables users to construct NCS elements using user-friendly sliders and text inputs, offering precise control over the truth, indeterminacy, and falsehood intervals. Robust error handling ensures that invalid or out-of-bound inputs are detected and avoided, preventing unexpected behavior during operations.

The tool supports two separate sets - NCS A and NCS B, allowing users to perform side-by-side comparisons and execute a range of fundamental unary and binary operations. This design facilitates experimentation, visualization, and validation of theoretical properties of NCS in an accessible and reproducible manner.

5.2|Implementation and Tech Stack

Frontend: The tool's interface is implemented using Streamlit, an open-source Python library for building interactive and user-friendly web applications. Streamlit widgets such as sliders, text inputs, buttons, and selectors enable users to intuitively define and modify NCS parameters in real time, with immediate visual feedback. This interactive design enhances usability by allowing users to experiment with different configurations and observe the results of operations dynamically, without requiring extensive programming knowledge.

Backend: Python is chosen for the backend due to its clear and easy-to-understand syntax, as well as its extensive library support. The core functionality of the tool is built using Abstract Data Types (ADTs) to represent and manage Neutrosophic Cubic Sets, Neutrosophic Cubic Numbers, Neutrosophic Sets, and Interval Neutrosophic Sets. These custom classes encapsulate the underlying logic for defining elements, performing various operations, and ensuring that the mathematical constraints of neutrosophic logic are rigorously maintained throughout execution.

5.3|Logical flow and Functional execution of the NCS Tool

The execution flow of the proposed tool, depicted in Fig. 1, outlines the key stages involved in the construction and manipulation of Neutrosophic Cubic Sets. Users begin by entering the parameters for NCS A and NCS B, defined by interval-valued truth, indeterminacy, and falsity components, along with their corresponding single-valued counterparts. These inputs are used to instantiate Neutrosophic Cubic Numbers, which are subsequently added to their respective NCS objects.

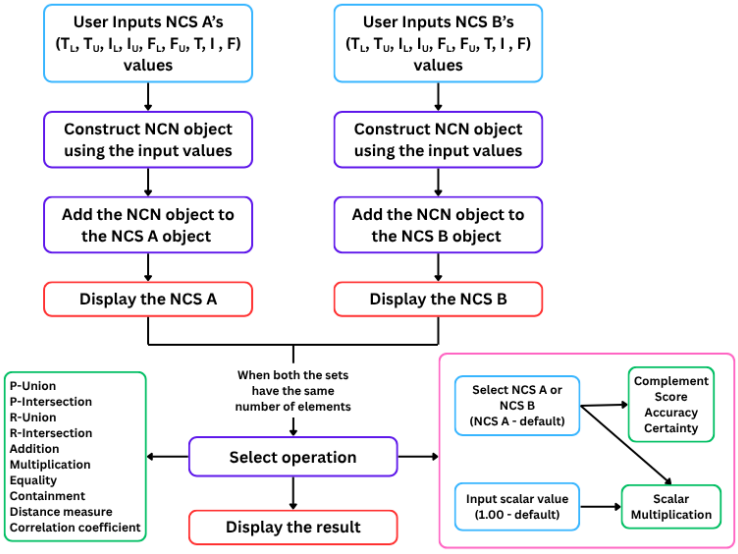

User Inputs NCS A's (T_L, T_U, I_L, I_U, F_L, F_U, T, I , F) values
User Inputs NCS B's (T_L, T_U, I_L, I_U, F_L, F_U, T, I , F) values
Construct NCN object using the input values
Construct NCN object using the input values
Add the NCN object to the NCS A object
Add the NCN object to the NCS B object
Display the NCS A
Display the NCS B
When both the sets have the same number of elements
P-Union
P-Intersection
R-Union
R-Intersection
Addition
Multiplication
Equality
Containment
Distance measure
Correlation coefficient
Select operation
Display the result
Select NCS A or NCS B (NCS A - default)
Complement
Score
Accuracy
Certainty
Input scalar value (1.00 - default)
Scalar Multiplication

Fig. 1. Logical flow of the proposed tool for Neutrosophic Cubic Sets.

Upon ensuring that both sets contain an equal number of elements, the system enables a suite of operations. Binary operations include P-union, R-union, P-intersection, R-intersection, addition and multiplication and scalar multiplication, as well as comparisons like equality and containment. Unary operations, applicable to either NCS A or B, include complement, score, accuracy, certainty. The outcome of each operation is computed and rendered dynamically, supported by Streamlit's reactive interface, allowing for immediate feedback and interactive exploration of the results, as illustrated in Fig. 2.

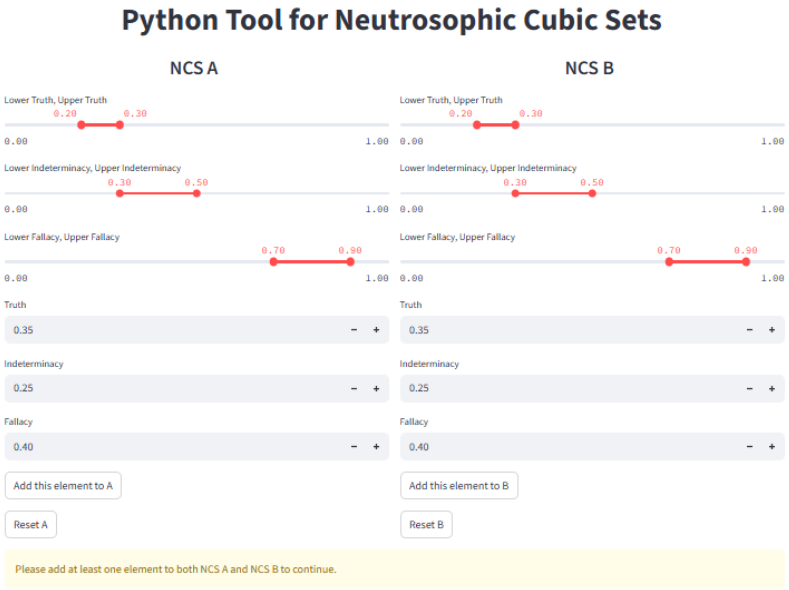


Fig. 2. Interactive tool for defining two Neutrosophic Cubic Sets through dynamic sliders and structured input fields.

To demonstrate the functionality of the tool, consider two Neutrosophic Cubic sets A and B given by,

$$A = \Big\{\langle[0.2,0.3],[0.3,0.5],[0.3,0.5]\rangle,\langle 0.35,0.25,0.4\rangle,\ \langle[0.4,0.7],[0.1,0.4],[0.2,0.4]\rangle,\langle 0.35,0.5,0.3\rangle,\ \langle[0.6,0.9],[0.0,0.2],[0.3,0.4]\rangle,\langle 0.5,0.6,0.55\rangle\Big\}$$

$$B = \Big\{\langle[0.3,0.7],[0.3,0.5],[0.1,0.5]\rangle,\langle 0.25,0.25,0.6\rangle,\ \langle[0.5,0.8],[0.5,0.6],[0.2,0.5]\rangle,\langle 0.45,0.3,0.3\rangle,\ \langle[0.4,0.9],[0.4,0.7],[0.3,0.5]\rangle,\langle 0.35,0.1,0.45\rangle\Big\}$$

According to Definition 0.4, the maximum values of truth, indeterminacy, and falsity-membership are retained in this operation, and the result of the P-union, as shown below, is illustrated in Fig. 3.

$$A \cup_{\mathrm{P}} B = \Big\{\langle[0.3,0.7],[0.3,0.5],[0.1,0.5]\rangle,\langle 0.35,0.25,0.6\rangle,\ \langle[0.5,0.8],[0.5,0.6],[0.2,0.5]\rangle,\langle 0.45,0.3,0.3\rangle,\ \langle[0.6,0.9],[0.4,0.7],[0.3,0.5]\rangle,\langle 0.5,0.6,0.55\rangle\Big\}$$

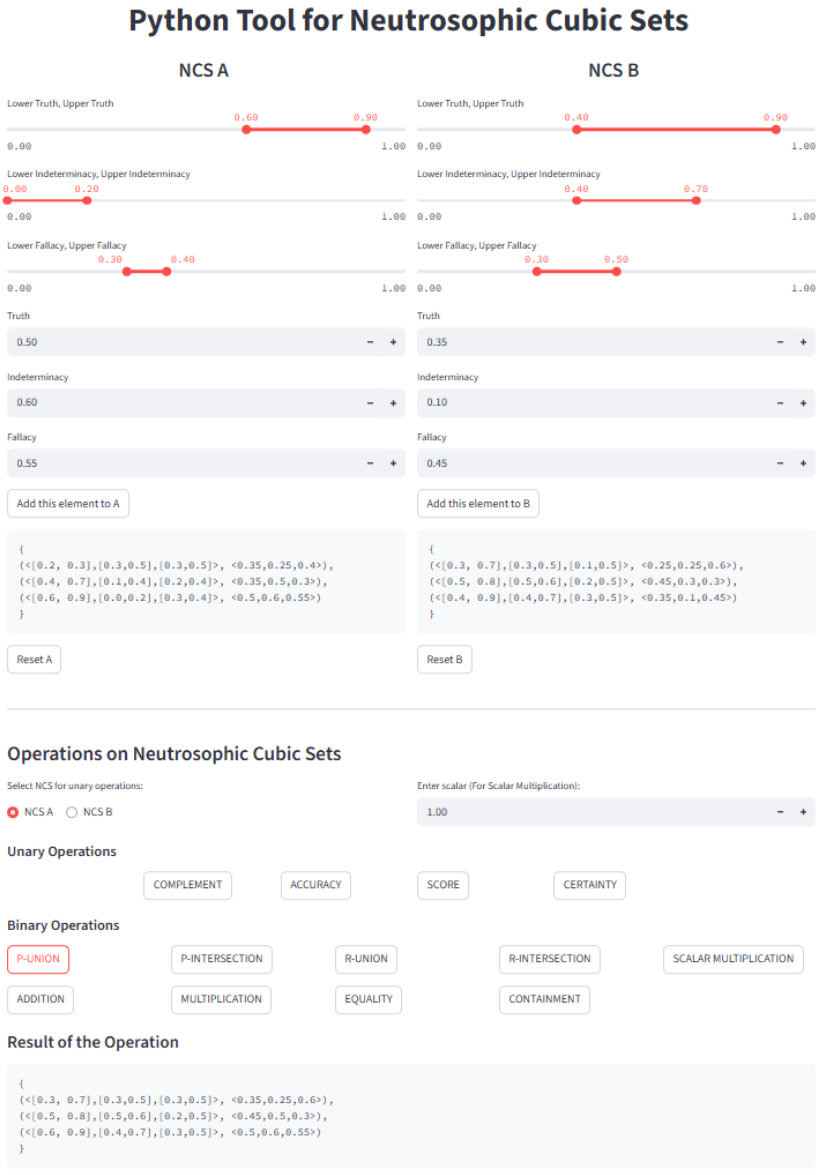


Fig. 3. Interface illustrating the execution and result of the P-union operation applied to two defined Neutrosophic Cubic Sets A and B.

Fig. 4 highlights the execution and resulting values of the scalar multiplication operation when applied to B with the scalar value 5. According to Definition 0.12, this operation models the effect of scaling certainty and uncertainty in a Neutrosophic Cubic Set by adjusting truth upward and compressing indeterminacy and falsity using exponential transformation.

$$\begin{aligned} 5 * B \quad = \quad & \Big\{ \langle [0.83, 1.0], [0.0, 0.03], [0.0, 0.03] \rangle, \langle 0.76, 0.0, 0.08 \rangle, \\ & \langle [0.97, 1.0], [0.03, 0.08], [0.0, 0.03] \rangle, \langle 0.95, 0.0, 0.0 \rangle, \\ & \langle [0.92, 1.0], [0.01, 0.17], [0.0, 0.03] \rangle, \langle 0.88, 0.0, 0.02 \rangle \Big\} \end{aligned}$$

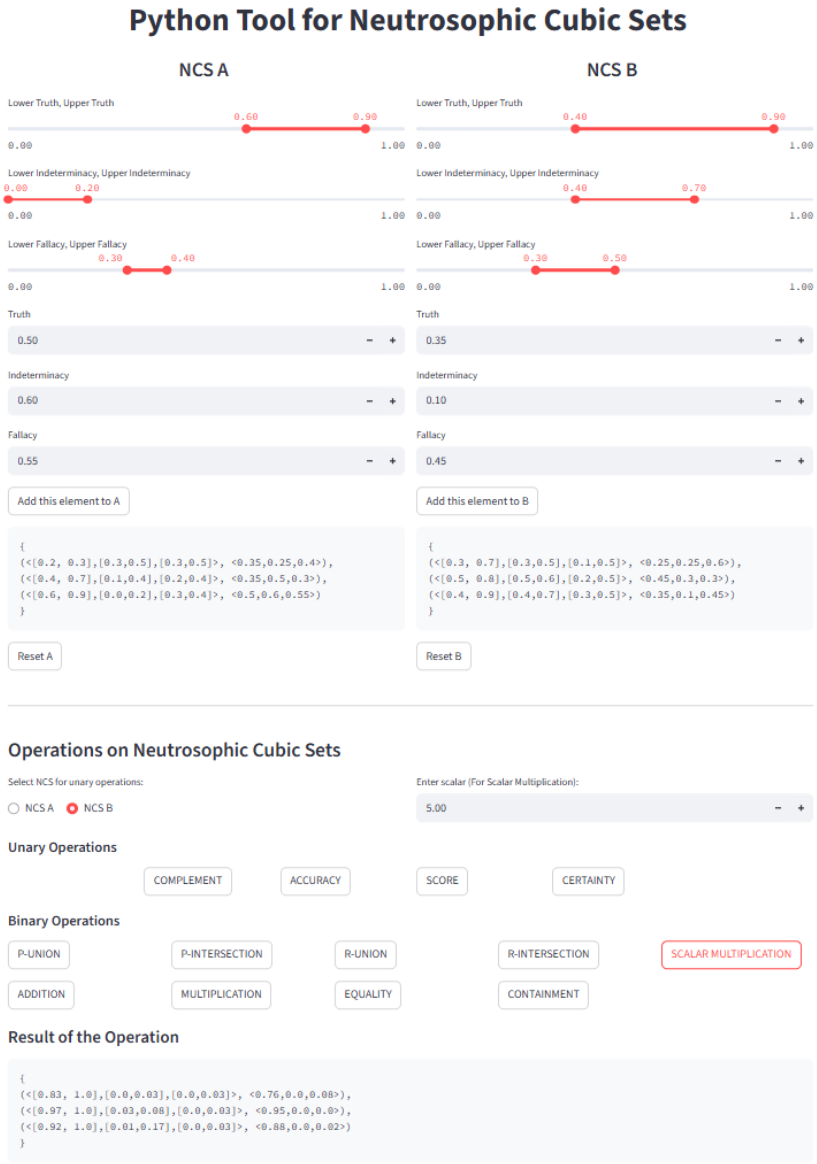


Fig. 4. Interface illustrating the execution and result of the scalar multiplication operation on Neutrosophic Cubic Set B with scalar $k = 5$.

Definition 0.15 gives the formula for computing the accuracy of a Neutrosophic Cubic Set. Fig. 5 illustrates the result of the accuracy operation applied to Neutrosophic Cubic Set B, which produces elementwise values, aligned with each component in the set.

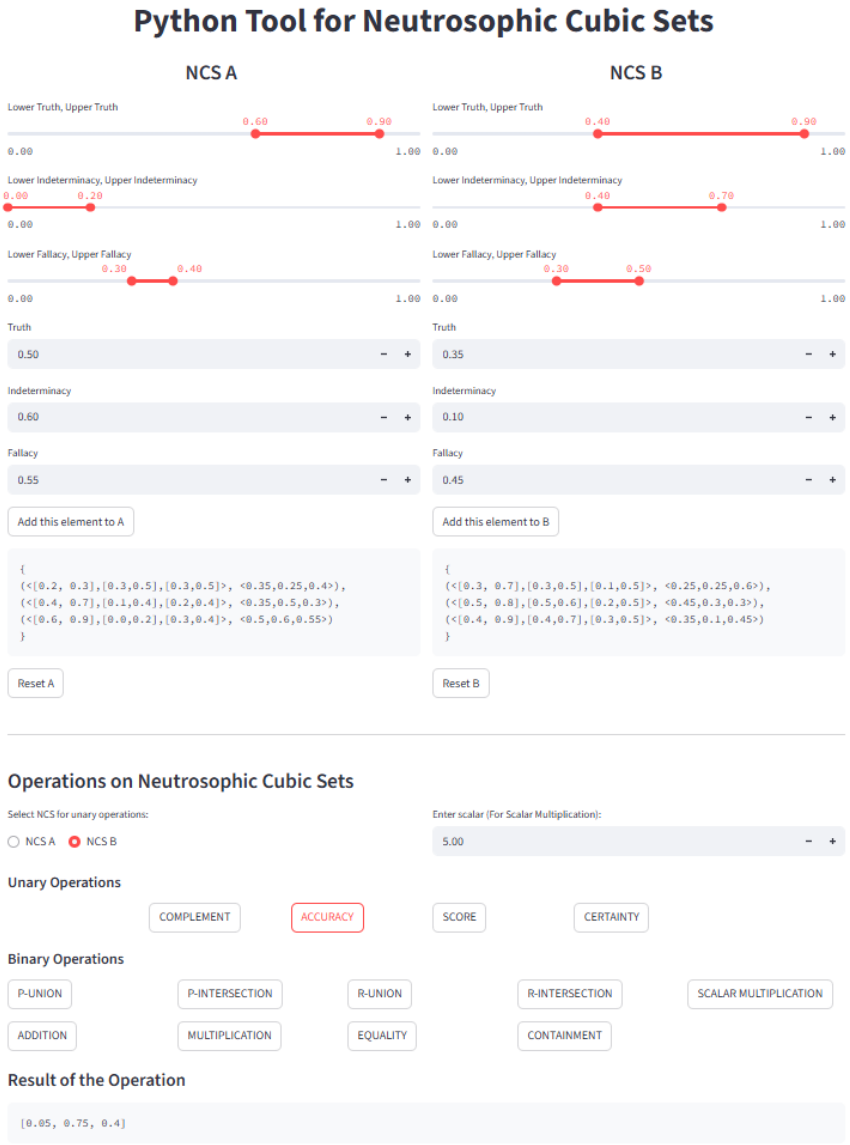


Fig. 5. Interface illustrating the computation of the accuracy of Neutrosophic Cubic Set B.

5.4|Operational Efficiency of the Tool

The runtime performance of the proposed Neutrosophic Cubic Set tool is largely determined by the number of elements n in each set and the element-wise operations defined over Neutrosophic Cubic Numbers. In the implementation, each NCN element consists of nine components in total. These are handled using Python classes (NCN and NCS) with efficient attribute-level access.

Unary operations are applied independently to each NCN in the selected set using single-pass iterations. As implemented, these follow a linear time complexity of $\mathcal{O}(n)$, where n is the number of NCNs in the selected NCS.

Binary operations are only allowed when NCS A and B have equal cardinality. Each operation iterates once through all pairs of NCNs at corresponding positions in the two sets, resulting in a time complexity of $\mathcal{O}(n)$.

In practice, the interactive frontend introduces negligible overhead, and the tool maintains real-time performance during typical usage scenarios, making it well-suited for exploratory analysis.

6|Neutrosophic Cubic Set based Image Enhancement

Image enhancement plays a crucial role in improving the visual quality and extractable information from an image. In this section, the concept of Neutrosophic Cubic Sets is applied to grayscale images to emphasize fine details and local structures. Each pixel is transformed into a Neutrosophic Cubic Number, which encodes its truth, indeterminacy, and falsity values based on local neighborhood information. The procedure used for converting a grayscale image into its corresponding Neutrosophic Cubic Set is detailed in Algorithm 1.

Algorithm 1 Conversion of Grayscale Image to Neutrosophic Cubic Set

Require: Grayscale image I of size $H \times W$, threshold parameter ε
Ensure: Neutrosophic Cubic Set NCS

Pad I by 1 pixel on all sides using edge replication
for each pixel (i, j) in I **do**
    Extract the 3×3 local neighborhood centered at (i, j)
    Compute the local mean $\mu = \text{mean}(3 \times 3 \text{ region})/255$
    Normalize pixel value: $T = I[i, j]/255$
    Compute: $I = 1 - |T - \mu|, \quad F = 1 - T$
    Compute intervals:
    $T_l = \max(0, T - \varepsilon), \quad T_u = \min(1, T + \varepsilon)$
    $I_l = \max(0, I - \varepsilon), \quad I_u = \min(1, I + \varepsilon)$
    $F_l = \max(0, F - \varepsilon), \quad F_u = \min(1, F + \varepsilon)$
    Create NCN using these values and add it to the NCS
end for
return NCS

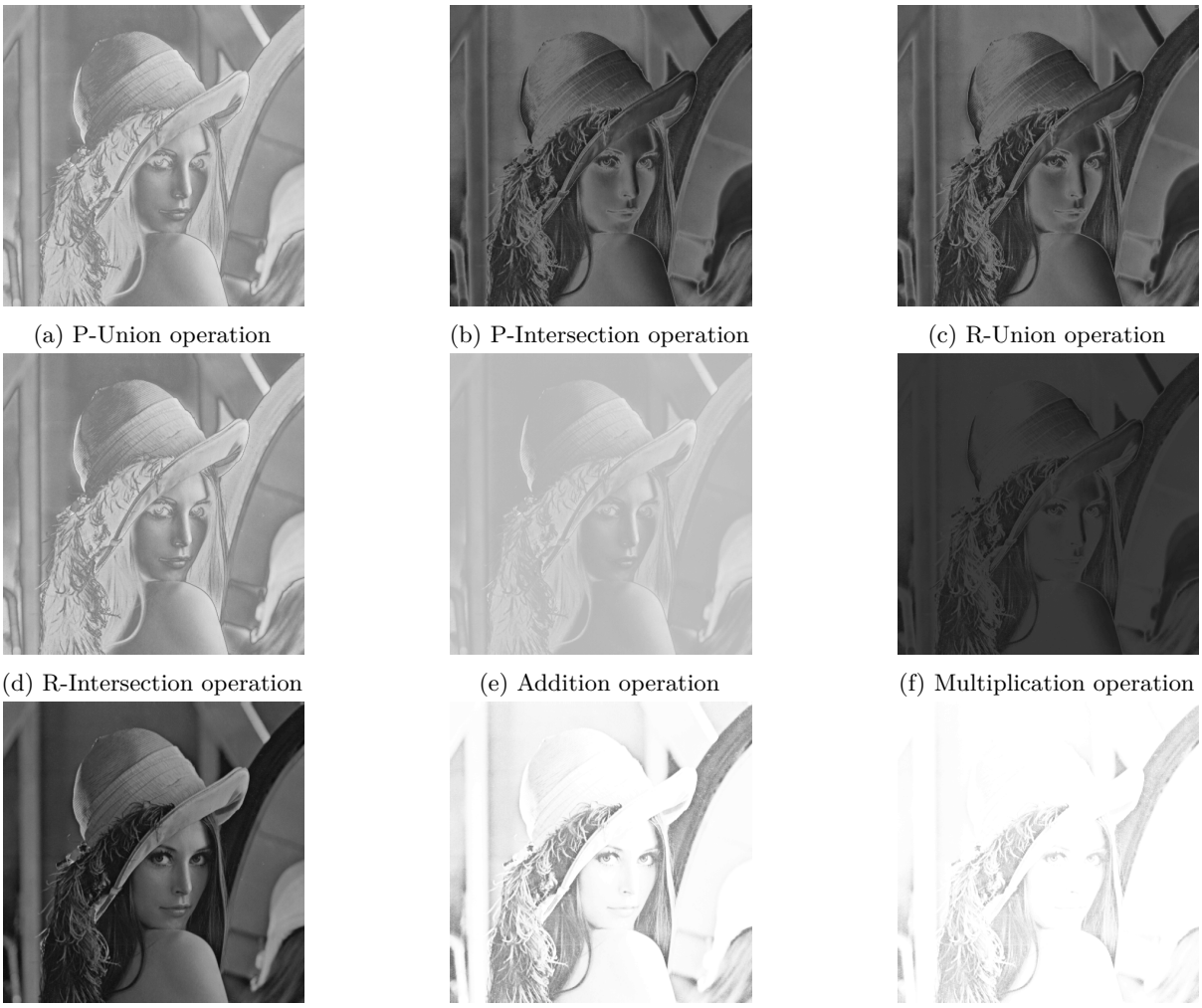
(a) Original Lenna image (b) Grayscale version of Lenna image (c) Reconstructed Lenna image

Fig. 6. Comparison of the original, grayscale and reconstructed images. The reconstructed image is obtained by encoding the grayscale image into a NCS and decoding it back to pixel values.

Fig. 6 presents the original Lenna image, its grayscale version, and the reconstructed image (via NCS transformation and inversion), revealing visibly enhanced features. Fig. 7 shows the Lenna image and its complement obtained through the complement operation. The inferences drawn from the remaining Neutrosophic Cubic Set operations are based on the grayscale Lenna image and its complement.

(a) Grayscale version of Lenna image (b) Complement of grayscale image

Fig. 7. Comparison between the grayscale Lenna image and its complement obtained using the complement operation on the Neutrosophic Cubic Set representation.



(a) P-Union operation (b) P-Intersection operation (c) R-Union operation

(d) R-Intersection operation (e) Addition operation (f) Multiplication operation

(g) Scalar Multiplication (0.5) (h) Scalar Multiplication (5) (i) Scalar Multiplication (10)

Fig. 8. Visual results of various Neutrosophic Cubic Set operations applied to the grayscale Lenna image and its complement, including union, intersection, addition, multiplication, and scalar multiplication with factors 0.5, 5, and 10.

The visual results in Fig. 8 demonstrate how various Neutrosophic Cubic Set operations can be used to enhance image features. The P-union (Fig. 8a) and R-intersection (Fig. 8d) operations results show brighter images, illustrating how P-union operation combines information to increase overall intensity, producing slightly higher brightness than the R-intersection. In contrast, the outputs of P-intersection (Fig. 8b) and R-union (Fig. 8c) operations appear darker, indicating that these operations retain only the common membership, which helps reduce noise and uncertainty while preserving essential structural details. P-intersection preserves slightly more information than the R-union. The addition operation (Fig. 8e) significantly brightens the image by summing the sets, boosting contrast but potentially causing overexposure if not regulated. The multiplication output (Fig. 8f) darkens the image, demonstrating its ability to suppress noise by highlighting regions with strong agreement between the sets. Finally, the effect of scalar multiplication with factors 0.5, 5, and 10 (Figures 8g, 8h, and 8i) shows how scaling influences image intensity: a lower factor reduces brightness and contrast, while higher factors progressively increase both, illustrating how the scalar parameter can be adjusted to achieve the desired level of enhancement.

Table 1. Comparison of original and noisy Lenna images based on Score, Accuracy, and Certainty.

Image (Grayscale)	**Preview**	**Score**	**Accuracy**	**Certainty**
Original Lenna image		0.354	0.116	1.558
Lenna image with white stripes		0.352	0.102	1.551
Lenna image with salt and pepper noise		0.333	-0.083	1.459

From the results presented in Table 1, it can be inferred that the original image consistently shows higher values for score, accuracy, and certainty compared to the noisy variants. This indicates that these measures effectively reflect the level of distortion or quality degradation in an image. Consequently, they can serve as useful indicators in decision-making processes [2] where selecting images with higher quality and lower uncertainty is critical, such as in image preprocessing, enhancement, or classification tasks.

7|Limitations of the Study

While the proposed Python-based tool for Neutrosophic Cubic Sets provides a comprehensive and interactive environment for performing a wide range of operations, a few limitations remain that are worth acknowledging:

- **Domain-Specific Customization:** The tool offers general-purpose operations but does not currently support specialized tuning for domain-specific applications such as medical diagnosis, financial forecasting, or real-time decision systems.

- **Lack of Statistical Integration:** While score, accuracy, and certainty metrics are implemented, the tool does not yet support advanced statistical analysis modules that could help in automating pattern recognition or decision-making processes.

8|Conclusion

In this paper, essential definitions related to Neutrosophic Cubic Sets have been presented as an extension to those introduced in earlier research, along with Python implementation snippets and a user-friendly Python-based computational tool that enables the exploration of Neutrosophic Cubic Sets and their core operations. This tool provides researchers with an effective bridge between theoretical foundations and practical use, helping them better understand and apply the underlying theoretical constructs. Its functionality has been demonstrated through worked examples, and the impact of the defined operations has been illustrated using images, highlighting their usefulness for image enhancement tasks. Future research will focus on applying Neutrosophic Cubic Sets to specific domains such as image segmentation, pattern recognition, the processing and analysis of complex medical images, or other fields where managing uncertainty and indeterminacy is essential. The limitations identified in this study, such as the need for domain-specific customization and statistical integration, will also be systematically addressed in the future work.

Supplementary Material

1. Python implementation: GitHub Link
2. Interactive tool: Click here

Acknowledgments

The authors extend their sincere appreciation to the editors and anonymous reviewers for their meticulous evaluation and thoughtful feedback. Their critical insights and constructive recommendations significantly contributed to improving the clarity and overall quality of this manuscript.

Author Contributions

Data Availability

All data supporting the findings of this study are included within the manuscript. Hyperlinks to supplementary online resources have been provided where applicable.

Funding

The authors declare that no specific funding was received for the conduct of this research.

Conflicts of Interest

The authors declare that they have no competing interests that could have influenced the work presented in this manuscript.

References

[1] Broumi, S., Bakali, A., & Bahnasse, A. (2018). Neutrosophic sets: An overview. *Neutrosophic sets and systems*, 22, 7–15. https://doi.org/10.5281/zenodo.1237940

[2] Banerjee, D., Giri, B. C., Pramanik, S., & Smarandache, F. (2017). GRA for multi attribute decision making in neutrosophic cubic set environment. *Neutrosophic sets and systems*, *16*, 15–23. https://doi.org/10.5281/zenodo.570938

[3] Ali, M., Deli, I., & Smarandache, F. (2016). The theory of neutrosophic cubic sets and their applications in pattern recognition. *Journal of intelligent & fuzzy systems*, *30*(4), 1957–1963. http://dx.doi.org/10.3233/IFS-151906

[4] El-Ghareeb, H. A. (2019). Novel open-source Python neutrosophic package. *Neutrosophic sets and systems*, *26*, 73–84. https://doi.org/10.5281/zenodo.2631514

[5] Nordo, G., Jafari, S., Mehmood, A., & Basumatary, B. (2024). A Python framework for neutrosophic sets and mappings. *Neutrosophic sets and systems*, *65*, 199–236. https://doi.org/10.5281/zenodo.10858945

[6] Jun, Y. B., Smarandache, F., & Kim, C. S. (2017). Neutrosophic cubic sets. *New mathematics and natural computation*, *13*(1), 41–54. https://doi.org/10.1142/S1793005717500041

[7] ShaliniTharanya, J., Pricilla, M. T., & Cruz, R. A. (2024). Neutrosophic spherical cubic sets. *Neutrosophic sets and systems*, *63*(1), 45–59. https://digitalrepository.unm.edu/nss_journal/vol63/iss1/3/

[8] Darvesh, A., Naeem, K., & Shah, S. B. H. (2025). Scrutiny of neutrosophic cubic fuzzy data using Bonferroni weighted mean. *Decision making advances*, 3(1), 245–265. https://doi.org/10.31181/dma31202576

[9] Fujita, T. (2025). Some types of hyper-neutrosophic sets: Cubic, trapezoidal, and offset models. *Neutrosophic sets and systems*, *67*, 55–78. https://digitalrepository.unm.edu/math_fsp/785/

[10] Smarandache, F., Broumi, S., & Deli, I. (2023). Designing pentapartitioned neutrosophic cubic set aggregation operator-based air pollution decision-making model. *Complex & intelligent systems*, *9*, 1231–1247. https://doi.org/10.1007/s40747-023-00971-2

[11] Smarandache, F. (1998). *Neutrosophy: Neutrosophic probability, set, and logic: Analytic synthesis & synthetic analysis*. American Research Press. https://philpapers.org/rec/Smannp

[12] Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, *8*(3), 338–353. http://dx.doi.org/10.1016/S0019-9958(65)90241-X

[13] Zadeh, L. A. (1971). Similarity relations and fuzzy orderings. *Information sciences*, *3*(2), 177–200. https://doi.org/10.2307/2272014

[14] Turksen, I. B. (1986). Interval valued fuzzy sets based on normal forms. *Fuzzy sets and systems*, *20*(2), 191–210. http://dx.doi.org/10.1016/0165-0114(86)90077-1

[15] Jun, Y. B., Kim, C. S., & Yang, K. (2012). Cubic sets. *Annals of fuzzy mathematics and informatics*, *4*(1), 83–98. http://www.afmi.or.kr/papers/2012/Vol-04_No-01/AFMI-4-1(83-98)-J-111023.pdf

[16] Wang, H., Smarandache, F., Sunderraman, R., & Zhang, Y.-Q. (2005). Interval neutrosophic sets and logic: theory and applications in computing. *Infinite Study*. ISBN: 1-931233-91-8

[17] Haibin, W., Smarandache, F., & Sunderraman, R. Z. Y. Q. (2010). *Single valued neutrosophic sets*. https://digitalrepository.unm.edu/math_fsp/708/

[18] Ponnukumar, P. S., Francis Xavier, N. I., & Jaganathan, R. (2025). Stable plithogenic cubic sets. *Journal of fuzzy extension and applications*, *6*(2), 410–423. https://doi.org/10.22105/jfea.2025.449408.1422

[19] Bodur, S., Topal, S., Gürkan, H., & Edalatpanah, S. A. (2024). A novel neutrosophic likert scale analysis of perceptions of organizational distributive justice via a score function: a complete statistical study and symmetry evidence using real-life survey data. *Symmetry*, *16*(5), 598. http://dx.doi.org/10.3390/sym16050598

[20] Jennifer, J. S., & Sharmila, T. S. (2023). A neutrosophic set approach on chest X-rays for automatic lung infection detection. *Information technology and control*, *52*(1), 37–52. http://dx.doi.org/10.5755/j01.itc.52.1.31520

[21] Jennifer, J. S., Kalaivani, C. (2024). A neutrosophic approach for breast mass detection in digital mammogram images. *Neutrosophic sets and systems*, *73*(1), 11. https://fs.unm.edu/nss8/index.php/111/article/view/5018

[22] Kaur, G., & Garg, H. (2022). A new method for image processing using generalized linguistic neutrosophic cubic aggregation operator. *Complex & intelligent systems*, *8*(6), 4911–4937. http://dx.doi.org/10.1007/s40747-022-00718-5

[23] Ghanbari Talouki, A., Koochari, A., & Edalatpanah, S. A. (2025). Video completion in the presence of moving subjects based on segmentation using neutrosophic sets. *Big data and computing visions*, *5*(1), 1–10. https://doi.org/10.22105/bdcv.2024.477690.1188

[24] Alshikho, M., Jdid, M., & Broumi, S. (2023). Artificial intelligence and neutrosophic machine learning in the diagnosis and detection of COVID 19. *Journal prospects for applied mathematics and data analysis*, *1*(2), 17-27. http://dx.doi.org/10.54216/PAMDA.010202

[25] Amin, K. M., Shahin, A. I., & Guo, Y. (2016). A novel breast tumor classification algorithm using neutrosophic score features. *Measurement*, *81*, 210–220. http://dx.doi.org/10.1016/j.measurement.2015.12.013